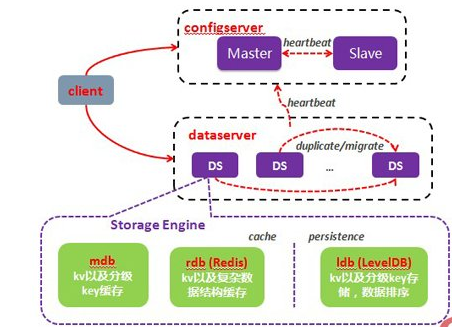
一、Tair技术概述

tair,是淘宝开源的一个高性能、分布式、可扩展、高可靠的NoSQL k/v存储系统. tair 分为持久化和非持久化两种使用方式. 非持久化的 tair 可以看成是一个分布式缓存. 持久化的 tair 将数据存放于磁盘中.

tair集群主要包括3个必选模块：configserver、dataserver和client，一个可选模块：invalidserver。通常情况下，一个集群中包含2台configserver及多台dataServer。两台configserver互为主备并通过维护和dataserver之间的心跳获知集群中存活可用的dataserver，构建数据在集群中的分布信息（对照表）。dataserver负责数据的存储，并按照configserver的指示完成数据的复制和迁移工作。client在启动的时候，从configserver获取数据分布信息，根据数据分布信息和相应的dataserver交互完成用户的请求。invalidserver主要负责对等集群的删除和隐藏操作，保证对等集群的数据一致。

从架构上看，configserver的角色类似于传统应用系统的中心节点，整个集群服务依赖于configserver的正常工作。但实际上相对来说，tair的configserver是非常轻量级的，当正在工作的服务器宕机的时候另外一台会在秒级别时间内自动接管。而且，如果出现两台服务器同时宕机的最恶劣情况，只要应用服务器没有新的变化， tair依然服务正常。而有了configserver这个中心节点，带来的好处就是应用在使用的时候只需要配置configserver的地址（现在可以直接配置Diamond key），而不需要知道内部节点的情况。

架构图：



二、服务器功能介绍

**ConfigServer**

介绍：**configserver维护分布式对照表，此表有一个version，当有dataserver删除与增加时，configserver会重新生成对照表，并同步给所有的dataserver,而客户端第一次访问时会从configserver中坊取对应的对象表信息并缓存，然后数据key与对应的dataserver通信，每次跟dataserver交互时，会比较对照表的版本号，如果有更新，则会重新请求configserver读取对应的对照表。**

功能：

* 通过维护和dataserver心跳来获知集群中存活节点的信息
* 根据存活节点的信息来构建数据在集群中的分布表。
* 提供数据分布表的查询服务。
* 调度dataserver之间的数据迁移、复制。

### DataServer的功能

### 介绍：Tair抽象了dataserver的存储引擎，底层可以基于memcache，LevelDb, redis等各种存储引擎保存数据，以实现持久化和非持久化功能。

### 功能：

* 提供存储引擎
* 接受client的put/get/remove等操作
* 执行数据迁移，复制等
* 插件：在接受请求的时候处理一些自定义功能
* 访问统计

### Client的功能

* 在应用端提供访问Tair集群的接口。
* 更新并缓存数据分布表和invalidserver地址等。
* LocalCache，避免过热数据访问影响tair集群服务。
* 命令流控

三、数据分布

分布式系统需要解决的一个重要问题便是决定数据在集群中的分布策略，好的分布策略应该能将数据均衡地分布到所有节点上，并且还应该能适应集群节点的变化。Tair采用的对照表方式较好地满足了这两点。对照表的行数是一个固定值，这个固定值应该远大于一个集群的物理机器数，由于对照表是需要和每个使用Tair的客户端同步的，所以不能太大，不然同步将带来较大的开销。生产环境中的行数一般为1023 。

对照表简介下面看对照表是怎么完成数据的分布功能的，为了方便，这里假设对照表的行数为6。最简单的对照表包含两列，第一列为hash值，第二列为负责该hash值对应数据的dataserver节点信息。比如有两个节点192.168.10.1和

192.168.10.2，那么对照表类似：

0       192.168.10.1

1       192.168.10.2

2       192.168.10.1

3       192.168.10.2

4       192.168.10.1

5       192.168.10.2

当客户端接收到请求后，将key的hash值和6取模，然后根据取模后的结果查找对照表。比如取模后的值为3，客户端将和192.168.10.2通信。

tair的分布式策略是通过对照表的方式，hash对应server的结点信息，然后key，hash后取模，以上方式可以很好的增加与删除结点，以及后面所涉及的备份等，考虑的比较全面。我们项目中在实现redis分布式缓存处理时，服务器结点信息做一致性哈希，然后数据key做hash选取对应的结点信息实现，此方式实现，数据能够均均分布，在增加与删除结点时，需要重新init hash的TreeMap，总体来说还是比较平滑。

四、对照表如何适应节点数量的变化

我们假设新增了一个节点——192.168.10.3，当configserver发现新增的节点后，会重新构建对照表。构建依据以下两个原则：

1. 数据在新表中均衡地分布到所有节点上。
2. 尽可能地保持现有的对照关系。

  更新之后的对照表如下所示：

|  |  |
| --- | --- |
| 0 | 192.168.10.1 |
| 1 | 192.168.10.2 |
| 2 | 192.168.10.1 |
| 3 | 192.168.10.2 |
| 4 | 192.168.10.3 |
| 5 | 192.168.10.3 |

  这里将原本由192.168.10.1负责的4和192.168.10.2负责的5交由新加入的节点192.168.10.3负责。

  如果是节点不可用，则相当于上述过程反过来，道理是一样的。

五、多备份的支持

Tair支持自定义的备份数，比如你可以设置数据备份为2，以提高数据的可靠性。对照表可以很方便地支持这个特性。我们以行数为6，两个节点为例，2个备份的对照表类似：

|  |  |  |
| --- | --- | --- |
| 0 | 192.168.10.1 | 192.168.10.2 |
| 1 | 192.168.10.2 | 192.168.10.1 |
| 2 | 192.168.10.1 | 192.168.10.2 |
| 3 | 192.168.10.2 | 192.168.10.1 |
| 4 | 192.168.10.1 | 192.168.10.2 |
| 5 | 192.168.10.2 | 192.168.10.1 |

  第二列为主节点的信息，第三列为辅节点信息。在Tair中，客户端的读写请求都是和主节点交互，所以如果一个节点不做主节点，那么它就退化成单纯的备份节点。因此，多备份的对照表在构建时需要尽可能保证各个节点作为主节点的个数相近。

  当有节点不可用时，如果是辅节点，那么configserver会重新为其指定一个辅节点，如果是持久化存储，还将复制数据到新的辅节点上。如果是主节点，那么configserver首先将辅节点提升为主节点，对外提供服务，并指定一个新的辅节点，确保数据的备份数。